DEFENDANT'S
EXHIBIT

DX-712

2:15-cv-00011-RSP

Using Kernel Hypervisors to Secure Applications

Terrence Mitchem, Raymond Lu, Richard O'Brien
Secure Computing Corporation
2675 Long Lake Road
Roseville, MN 55113 U.S.A
{mitchem, lu, obrien}@securecomputing.com

Abstract

This paper describes an approach for selectively controlling COTS components to provide robustness and security. Using the concept of a loadable module, "kernel hypervisors" have been implemented on a Linux kernel. These kernel hypervisors provide unbypassable security wrappers for application specific security requirements and can be used to provide replication services as well.

A framework has been developed based on a master kernel hypervisor, whose job is to coordinate installation and removal of individual client kernel hypervisors and to provide a means for management of these clients. The framework allows client kernel hypervisors to be stacked so that a variety of application specific policies can be implemented, each by means of its own kernel hypervisor. The hypervisors run in the kernel, but since they are loadable modules, they do not require that the kernel be modified.

Kernel hypervisors have a number of potential applications, including protecting user systems from malicious active content downloaded via a Web browser and wrapping servers and firewall services for limiting possible compromises.[1]

1 Introduction

A hypervisor is a layer of software, normally operating directly on a hardware platform, that implements the same instruction set as that hardware. They have traditionally been used to implement virtual machines[1]. This paper describes a similar concept, called a kernel hypervisor, that is implemented on top of an operating system kernel rather than on top of the hardware. These kernel hypervisors provide a set of "virtual" system calls for selected system components. They can be used to make a component more robust or to perform various security functions, such as application specific fine-grained access control and auditing of events.

Kernel hypervisors are loadable kernel modules that intercept system calls to perform pre-call and post-call processing. They can be used to provide an additional layer of fine-grained security control or to provide replication support. Their key features are that they can be set up to be unbypassable, since they are in the kernel, and they are easy to install, requiring no modification to the kernel or to the COTS applications that they are monitoring.

The goal of kernel hypervisors is to protect against malicious code or to provide some type of additional functionality. Hence, the operating assumption is that the user can be trusted. This assumption implies that kernel hypervisors are similar to virus protection programs in that a user must not specifically disable them. In fact, kernel hypervisors can be set up so that they cannot be disabled, but they do rely on proper user administration and use.

Section 2 of this paper presents a high level view of the kernel hypervisor architecture that we developed, and Section 3 gives a more detailed description of each component. Some possible uses for kernel hypervisors are discussed in Section 4.

2 Kernel hypervisor architecture

The kernel hypervisor architecture is illustrated in Figure 1. There are three main components:

- The master kernel hypervisor: manages the individual client kernel hypervisors that are currently loaded and provides a control facility allowing users to monitor and configure these client hypervisors.
- Specific client kernel hypervisors: provide application specific policy decision making and enforcement. Each client kernel hypervisor can

1. This work is sponsored by DARPA and Rome Lab under contract number: F30602-96-C-0338.

wrap one or many applications. User daemons that run in user space can also be developed to allow the client hypervisors to initiate actions in user space.

- Client hypervisor management module: provides an interface for communicating with and configuring the client kernel hypervisors from user space.

Kernel hypervisors are distinguished by the fact that they consist of *loadable kernel code* that is used to wrap specific applications. Other approaches have been used to provide wrappers for additional security or robustness including:

- Traditional hypervisors that replace the standard hardware interface with an interface that provides additional capabilities. This approach was used by Bressoud and Schneider[1] for building a replication hypervisor that intercepts, buffers, and distributes signals from outside the system.

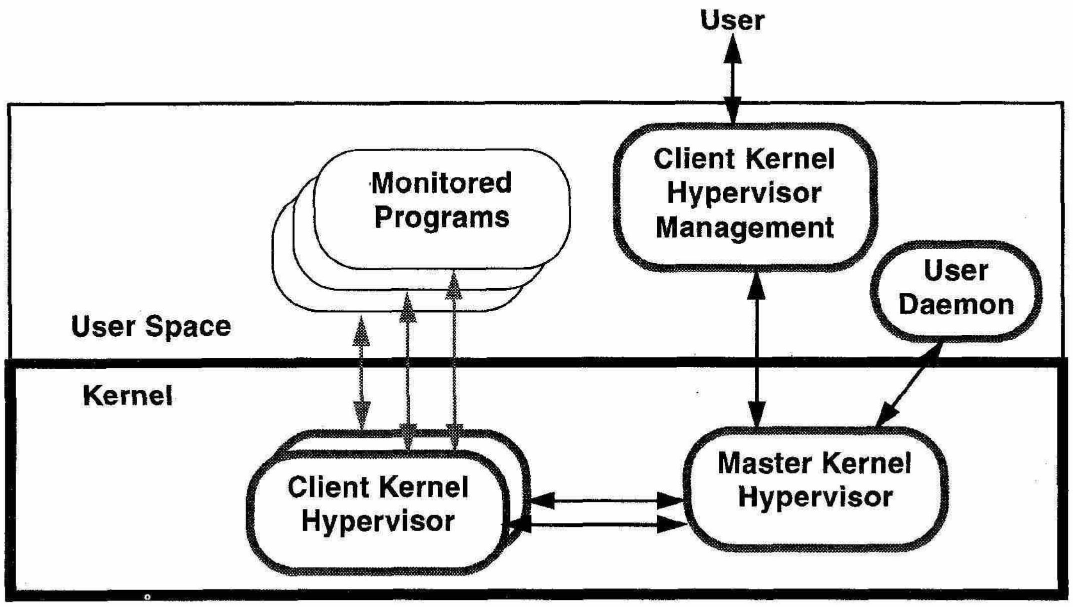
- Special libraries that include security functionality and that are linked with an application before it is run. This is the approach taken by SOCKS[2] where the client links in the SOCKS client library. SOCKS is intended for use with client/server TCP/IP applications, but the concept of linking in special libraries can be used for any type of application.

- Wrappers that make use of an operating system's debug functionality. This is the approach taken by the Berkeley group[3].

Our approach is most like the last, but it differs in that we place our code directly in the kernel and do not use the system debug functionality. Loadable kernel hypervisors have a number of benefits.

- They are unbypassable. Since the wrapper is implemented within the operating system kernel, malicious application code cannot avoid the wrapper by making direct calls to the operating system as could be done with code library wrappers. Any such calls will be intercepted and monitored.

- They do not require kernel modifications. All kernel hypervisor code is implemented in modules that are loadable within the kernel while



- Master Kernel Hypervisor: provides communications and control
- Client Kernel Hypervisors: provide application specific monitoring
- Management: provides user interface for configuration of client hypervisors

Figure 1. System Components

the system is running. There are no changes to kernel source code as is necessary with specialized secure operating systems.

- They are flexible. Kernel hypervisors can be used both to implement a variety of different types of security policies and to provide replication functionality. Some of the possible applications are discussed below. A key feature is that kernel hypervisors can be "stacked", so that modular security policies can be developed and implemented as needed. We are also currently investigating the implications of this stacking approach using composability theory. [4] [5]
- They are not platform specific. Kernel hypervisors can be used on any operating system that supports kernel loadable modules that have access to the system call data structure. This includes Linux, Solaris, and other modern Unix systems, as well as Windows NT.
- They can be used to wrap COTS software without any modification to the software (including, e.g., relinking). Kernel hypervisors can monitor the actions of the COTS software and react in a manner that is transparent to the COTS application, other than it may be denied access to certain system resources according to the policy being enforced by the kernel hypervisor.

3 Components of the system

Using the concept of a kernel loadable module, we have implemented kernel hypervisors on a Linux kernel. Linux was chosen as the initial platform because it supports loadable modules, the source code is free and easily available so results of our work will be accessible to other researchers and developers, and it is widely used as a Web server platform and hence provides a good target for our approach.

As Figure 2 below illustrates, Linux provides full support for kernel loadable modules[6]. Three system calls are supported that allow a privileged user to install and remove loadable modules and to list which modules are currently loaded. Once loaded, the kernel hypervisors in Linux run within kernel space and have full access to kernel data structures.

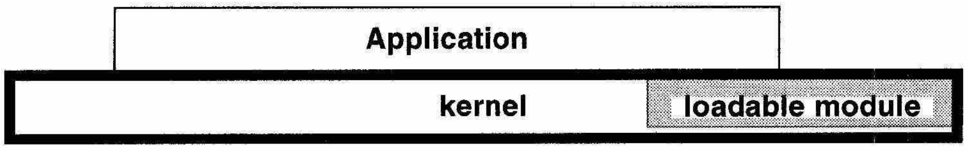
The remainder of this section discusses the three major components of our implemented kernel hypervisor system in more detail. These components are:

- the master hypervisor
- client kernel hypervisors
- kernel hypervisor management.

3.1 The master hypervisor framework

A framework has been developed based on a master kernel hypervisor, whose job is to coordinate installation and removal of individual client kernel hypervisors and to provide a means for management of these clients. The framework allows client kernel hypervisors to be stacked so that a variety of application specific policies can be implemented, each by means of its own kernel hypervisor. Each hypervisor runs in the kernel as a separate loadable module.

Figure 3 illustrates the framework. The master hypervisor is loaded before any client kernel hypervisors. A special application programming interface (API) has been defined that allows client hypervisors to register and unregister themselves with the master hypervisor and to identify which system calls they need to monitor. The master hypervisor keeps track of all currently registered client hypervisors and of the particular system calls that each client hypervisor is monitoring. When a client hypervisor module is removed via the rmmod call, it is the



Loadable modules operate within kernel space and have access to all kernel data structures
Loadable modules can be loaded/unloaded without any modifications to the running system
Commands:

lsmod	- list modules currently loaded
insmod *moduleName*	- load the module named *moduleName*
rmmod *moduleName*	- unload the module named *moduleName*

Figure 2. Linux Loadable Modules

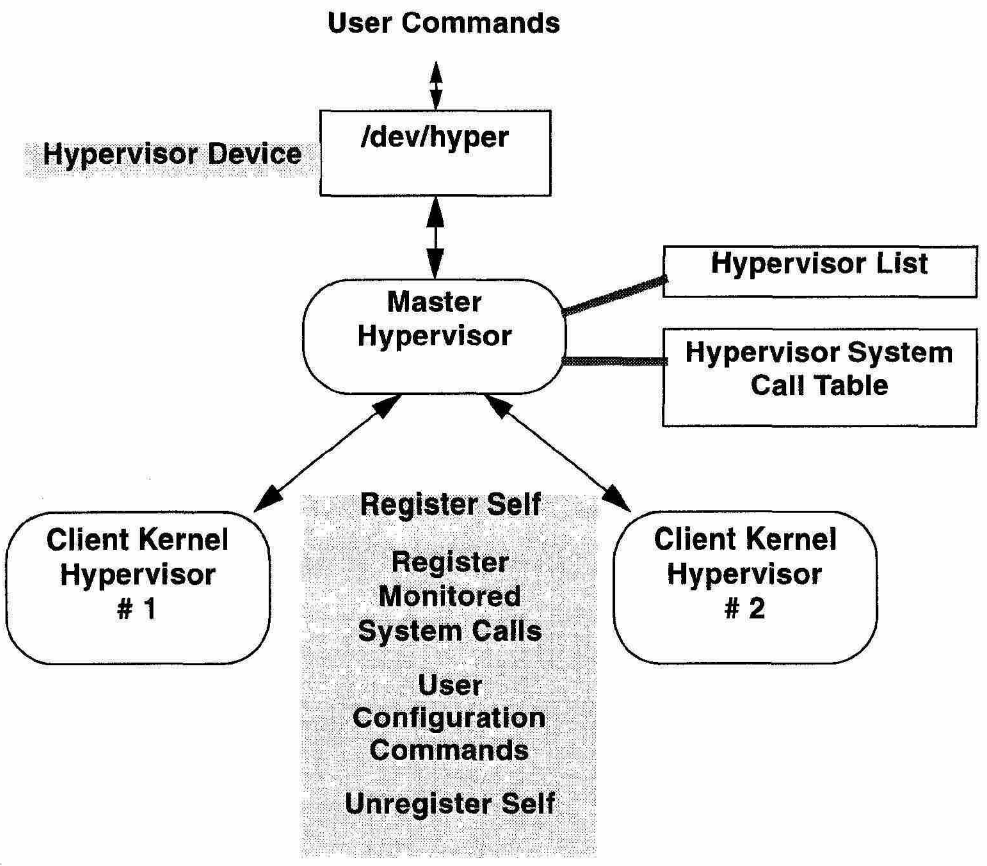
responsibility of the client hypervisor to unregister itself with the master hypervisor.

A special device, /dev/hyper, has been defined for communication between user space and kernel hypervisors. The master hypervisor acts as the device driver for this device. The device allows privileged users to dynamically update the configuration information for a kernel hypervisor, including updating the security policy that the hypervisor enforces. Which users are privileged can vary for different hypervisors, so that configuration can be tightly controlled. It also provides a mechanism that kernel hypervisors can use to communicate with user space daemons. Such daemons, for example, could be used to provide additional audit capabilities or replication services.

The master hypervisor provides special wrapper code for use with any system call that is being monitored. The actual monitoring of system calls is performed by redirecting the links in the kernel system call table to point to system call wrappers that the master hypervisor provides. This redirection of links is the only modification to the kernel that is performed and is done by the master hypervisor only on system calls being monitored. The wrapper code is invoked when the system call is made and performs the processing illustrated in Figure 4. Each system call has its own wrapper that follows the algorithm:

- For each client kernel hypervisor monitoring this call, initiate that client's pre-call processing.
- Call the standard system call processing.
- For each client kernel hypervisor monitoring this call, initiate that client's post-call processing.

Pre-call and post-call processing is used to enforce the client hypervisor's particular security policy or to initiate


User Commands
Hypervisor Device
/dev/hyper
Hypervisor List
Master Hypervisor
Hypervisor System Call Table
Register Self
Register Monitored System Calls
User Configuration Commands
Unregister Self
Client Kernel Hypervisor # 1
Client Kernel Hypervisor # 2

- Maintains hypervisor list
- Maintains hypervisor system call table
- Device driver for /dev/hyper

Figure 3. Master Hypervisor Functions

other actions by the client hypervisor. This processing could include additional auditing of system calls, including call parameters and results; performing access checks and making access decisions for controlled resources that the hypervisor is protecting; modifying system call parameters; and passing information to user daemons.

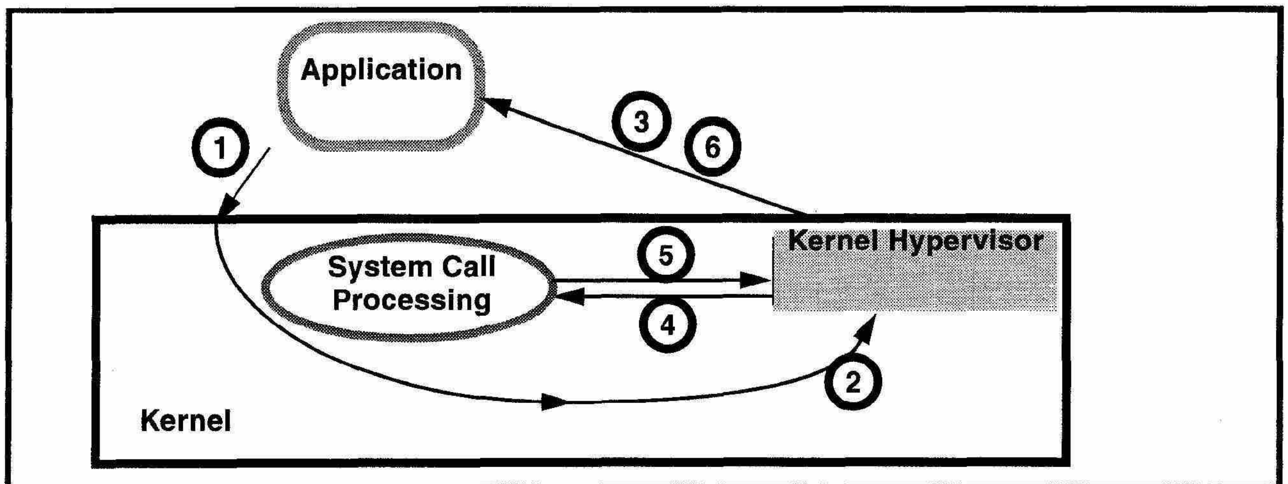
3.2 Client kernel hypervisors

Client kernel hypervisors are developed and loaded separately as needed. A single client kernel hypervisor can be designed to monitor just one specific application or a number of different applications. Client kernel hypervisors could also be used to enforce other types of policies independent of any application.

To illustrate the practicality of the kernel hypervisor concept, we developed a client kernel hypervisor for wrapping the Netscape browser. The goal of the Netscape hypervisor is to protect a user, browsing on the Internet, from downloading and executing malicious active content that might damage the user's system. The Netscape hypervisor accomplishes this by monitoring system calls made by the browser and enforcing a policy that only allowed certain resources to be accessed. In particular, the set of files that the browser can open for read, and read/write access is controlled so that the browser effectively operates within its own limited execution context. While this does not prevent malicious code from accessing and possibly damaging resources within this context, it does limit the damage that could be done to only these resources.

In the case of the Netscape browser, the context includes the user's .netscape directory as well as limited access to other libraries needed by the browser to execute. Most files on the system, however, are not accessible to the browser and so cannot be damaged. To ensure that applications started from the browser as the result of a download, e.g. a postscript viewer, are also controlled, the hypervisor keeps track of all descendents of the browser and enforces the same policy on them as on the browser.

The security policy that the Netscape hypervisor enforces is stated as a set of rules identifying which resources the browser is allowed to access and what permissions the browser has to the resource. If there is no rule that allows access to a resource, then the hypervisor



1	Initial System Call
2	System Call Redirected - Preprocessing Checks Made
3	Preprocessing Checks Fail - Return Failure
4	Preprocessing Checks Pass - Perform System Call Processing
5	Postprocessing Done
6	Return result of System Call Processing and Postprocessing

Figure 4. System Call Processing

refuses any requests for access to that resource. The format of the rules is:

<type> *<identifier>* *<permissions>*
where *<type>* is either a file, socket, or process
<identifier> is either a file/dir pathname, an IP address, or a process ID
and *<permissions>* depends on the type.

For our Netscape prototype, only rules for the *<file>* type are used. For these rules permissions are:
read, *write*, *read/write*, and *none*.

Certain conventions are used to simplify the statement of the rules. If an *<identifier>* is a file directory, then access to all files in that directory and all subdirectories is governed by the rule for the *<identifier>*, unless this rule is specifically overridden. Rules can be overridden by stating another, more specific, rule. For example, if a rule allows *read* access to all files and subdirectories of the directory /etc, then you can prevent users from accessing the file /etc/passwd by including a rule for this file with a permission of *none*. A graphical user interface tool for specifying the rules that define a given policy has also been developed. Also, based on our experience with the Netscape kernel hypervisor, a library of functions has been developed for use in implementing other kernel hypervisors.

The only performance penalty from using the Netscape hypervisor occurs when the application makes a system call, such as to open /create a file or to create another process, that is being monitored. Our performance tests have measured this penalty at approximately 10-30% on the particular system calls being monitored. It should be noted, however, that most system calls, such as read/writes to a file, do not need to be monitored, so that the overall penalty is minimized. There is no user noticeable performance penalty from running the hypervisor.

3.3 Kernel hypervisor management

The kernel hypervisor management component allows a user to obtain information on client hypervisors that are currently loaded and to reconfigure them, if needed. (As noted earlier, hypervisors are loaded and removed through standard Linux system calls.)

All hypervisor management is done via the communication through the /dev/hyper device. The master hypervisor responds to requests for a list of all currently installed client hypervisors and their current identifiers. These identifiers are used to direct requests to specific kernel hypervisors.

The management functions available for the Netscape hypervisor include:

- the ability to list the current rules that are being enforced
- the ability to clear the current rule set and load a new one
- the ability to change the log level that the hypervisor uses to determine what detail of logging it should do.

4 Possible applications

Kernel hypervisors can be used in a variety of ways to enhance the security and robustness of a system. This section discusses some of the types of uses followed by some specific applications.

4.1 Types of uses

Auditing: In their simplest form, kernel hypervisors can provide an audit and monitoring functionality that merely records additional information about the system resources that an application is accessing. Such audit hypervisors are also useful for determining what system resources an application accesses during its normal processing.

Fine-grained access control: The most compelling use of kernel hypervisors is to provide dynamic, fine-grained access control to various system resources such as files, network sockets and processes. The type of control possible is what differentiates this method of wrapping applications from standard access control list methods that perform control based on a user attribute. Kernel hypervisor security policies can be based on users, but can also be based directly on the application. For example, the Netscape kernel hypervisor that we implemented only monitored the Netscape application as well as any other applications started from within Netscape. Users had additional privileges to access their own .netscape directories, but could not override the restrictions in the Netscape hypervisor security database. These restrictions apply even to the root user. (In fact, an additional restriction imposed by the Netscape hypervisor was that the root user could not even run the Netscape browser.)

Label-based access control: While the rule sets described in Section 3.2 can be used to implement most simple policies, it would also be possible to use kernel hypervisors to implement more sophisticated label-based policies, such as a multilevel secure (MLS) policy or a type enforcement policy. To be able to implement these policies, a kernel hypervisor would need a way to label system resources. This might be done by creating and maintaining its own list of labelled names, or by piggy-backing the labels on system data structures that have available space. Because of the flexibility of the kernel hypervisor, these label-based policies could be applied to only those portions of the system that required labels. And policies could be quickly changed, if needed, to adapt to the current operating environment.

Replication: Kernel hypervisors can also be used to provide replication features by duplicating system calls that modify system resources, such as files. We have prototyped one approach for doing this by using a user daemon to replay file access requests that are intercepted by a kernel hypervisor. The replayed requests effectively duplicate that portion of the file system that is being monitored.

4.2 Specific applications

Specific applications that kernel hypervisors can be used for include:

- Wrapping web browsers to protect the user from downloaded malicious active content.
  This has already been described in the section on the Netscape hypervisor. The most interesting point is that any applications that are started automatically from the browser, as well as any plug-ins, are also subject to the same restrictions that are on the browser. As noted earlier, this limits the damage that malicious active content can do to only those areas of the system that the user allows the browser to access.
- Wrapping web servers to protect against malicious attacks.
  Kernel hypervisors can be used to ensure that if the web server is overrun the damage is limited. In particular, it can ensure that files being served by the web server are not modified inappropriately. Also, because they are often easy to overrun, CGI scripts are sometimes attacked by malicious users on the web. In particular, if a CGI script has the ability to connect to an internal system, then an attacker might be able to compromise this CGI script to launch an attack. By limiting which ports on an internal system a CGI script can connect to, a kernel hypervisor can limit the attacker to only services on those ports.
- Wrapping services and proxies on an application gateway.
  Services, like Sendmail, often have unknown bugs that are only discovered when someone uses them to attack the system on which the service is running. Once again, kernel hypervisors provide a way of limiting the damage that such a compromise can cause.
- Adding new security features to a system.
  Kernel hypervisors can be used to add security features, such as security levels, roles, or domains and types, to a system without requiring any additional modifications to the system. In fact, a special kernel security hypervisor could be implemented that performs all of the required security checks for any other kernel hypervisor.

This is only a partial list of the possible security uses of kernel hypervisors. Any number of applications could benefit from the additional security that they can provide.

5 Conclusion

Kernel hypervisors are an approach to wrapping applications to provide additional security that has a number of advantages over other approaches. Because they reside in the kernel, they cannot be bypassed. Because they are implemented as loadable modules, they do not require any modification to the kernel. Because they do not require modification to an application, they can be used to dynamically wrap processes that are started from other processes that have already been wrapped. Because they can be easily configured, the policy that they enforce can be dynamically modified as needed. By limiting the amount of damage malicious software can do, kernel hypervisors provide an approach to protecting one's system against current and future threats that may still be unknown.

References

[1] Thomas Bressoud and Fred Schneider. Hypervisor-based Fault-Tolerance. *Proceedings of the 15th ACM Symposium on Operating System Principles*. December, 1995. ACM Press.

[2] M. Leech, et al. RFC 1928: SOCKS Protocol Version 5. March 1996.

[3] Ian Goldberg, David Wagner, Randi Thomas and Eric Brewer. A Secure Environment for Untrusted Helper Applications. *Proceedings of the 6th USENIX Security Symposium*. July, 1996.

[4] Martin Abadi and Leslie Lamport. Conjoining Specifications. *ACM Transactions on Programming Languages and Systems* 17, 3. May 1995.

[5] Todd Fine. A Framework for Composability. *Proceedings of the Eleventh Annual Conference on Computer Assurance (COMPASS 96)*. June, 1996.

[6] Bjorn Ekwall and Jacques Gelinas. Linux Modules 2.1.3 Documentation. June, 1996. Distributed with Linux source.